Università degli Studi di Napoli
Federico II

Elaborato di

Progettazione e Sviluppo di Sistemi Software

No Spoiler

Sistema anti-food waste

Autori:

Francesco Maria Papulino	-	M63001009
Francesco Pietrantonio	-	M63001092
Giuseppe Ruggiero	-	M63000976
Martina Russo	-	000118117

Indice

Capitolo 1

Introduzione: Avviare il Progetto

- Scaricare i file dalla repository.
- Aprire MySqyl, creare il database con username "root" e password "root" utilizzando lo script ***costruisci_popola_db***. Se necessario, impostare il fuso orario corretto per MySQL.
- Aprire 5 finestre di terminale.
- Avviare i file *.jar* nell'ordine: ***Server.jar*** con il comando ***java -jar Server.jar***, ***Payment.jar***, ***Delivery.jar***.
- Avviare i file ***Cliente_MarioRossi.jar*** e ***Cliente_MarioBiondi.jar*** per visualizzare l'interfaccia grafica per due clienti differenti.

Capitolo 2

Inception

La fase di *inception*, o ideazione, serve a creare una visione iniziale comune tra gli stakeholders per il progetto che si andrà a sviluppare. Innanzitutto si procede con la definizione informale dei requisiti del progetto, almeno di quelli fondamentali, a mezzo di un cosiddetto *workshop*. Quest'ultimo ha quindi come output un primo documento informale di specifica.

2.1 Specifica informale dei requisiti

Alla fine del workshop sui requisiti l'analista produce il documento di specifica informale che è riportato in seguito. Tramite esso si procederà poi alla formalizzazione dei primi casi d'uso.

Specifica Informale dei Requisiti

Si vuole realizzare un'applicazione Java per la gestione di un sistema di compravendita di generi alimentari prossimi alla scadenza, al fine di ridurne lo spreco. I supermercati della zona che aderiranno al servizio inseriranno dei prodotti e gli utenti potranno visualizzarli ed acquistarli a prezzo ridotto.

Quando un supermercato decide di aderire al servizio, un referente designato effettuerà la registrazione al fine di inserire i propri prodotti in scadenza e renderli disponibili all'acquisto. Il referente è identificato dal suo nome e cognome.

Ogni supermercato è caratterizzato da un nome, una catena di afferenza, un indirizzo geografico e dagli orari di servizio. Un supermercato può operare in modo normale oppure in modalità promozione. La modalità promozione prevede un ulteriore sconto applicato su tutti i prodotti disponibili per la distribuzione. Ogni supermercato ha un proprio catalogo.

Ogni prodotto è caratterizzato da un codice a barre, una breve descrizione, la data di scadenza, il prezzo e lo sconto applicato. Ogni prodotto afferisce ad un solo supermercato.

Un cliente effettua la registrazione inserendo un username, una password, un indirizzo ed il nome per la consegna. Ogni cliente, una volta effettuato il login, può ricercare un prodotto nel sistema mediante due modalità di ricerca. Il primo tipo prevede di ricercare un prodotto in tutti i supermercati registrati e ritorna la lista dei supermercati che ne hanno almeno uno disponibile; il secondo tipo permette di ricercare un determinato supermercato per nome, per catena o per indirizzo e ritorna una lista di supermercati che rispondono al testo inserito.

Dopo aver ricercato un supermercato è possibile visualizzarne il catalogo e aggiungere uno o più prodotti al carrello. Il cliente potrà effettuare nel corso del tempo vari ordini, ognuno contenente i prodotti presenti nel suo carrello corrente e disponibili in almeno un supermercato. E' possibile inoltre rimuovere i prodotti dal carrello.

Se è presente almeno un prodotto disponibile nel carrello, il cliente potrà procedere al pagamento. Il pagamento e la consegna degli ordini saranno responsabilità di servizi esterni, un servizio di pagamento ed un servizio di delivery. Il sistema notificherà indirettamente ai "rider" i supermercati da cui ritirare l'ordine e il luogo dove consegnarlo. All'utente sarà notificato l'esito del pagamento e i dettagli della consegna.

Il referente di un supermercato può aggiornare la lista dei prodotti disponibili: è possibile aggiungere un prodotto alla lista oppure rimuovere un prodotto presente in lista. I prodotti inseriti all'interno del sistema da parte del referente dovranno avere una scadenza massima di due giorni dal momento in cui sono stati inseriti e saranno automaticamente rimossi dal sistema alla scadenza. Il supermercato sarà notificato nel caso in cui il prodotto risultasse invenduto e sarà aggiornata la lista dei prodotti disponibili.

2.2 Casi d'Uso

In seguito alla stesura del documento precedente si delineano gli attori che interagiranno con il sistema e i relativi casi d'uso. Nel nostro caso gli attori primari che interagiranno con il sistema saranno i ***clienti*** che acquisteranno prodotti attraverso l'applicazione ed i ***referenti***, impiegati dei vari supermercati che hanno il compito

di gestire i prodotti messi in vendita. L'applicazione avrà poi interazioni con un ***sistema di pagamento*** ed un ***sistema di delivery***, entrambi esterni. Di seguito sono mostrati la tabella attore-obiettivo ed il diagramma dei casi d'uso.

Tabella Attore-Obiettivo

Attore	*Obiettivo*	*Attore*	*Obiettivo*
Cliente	Cercare supermercati che vendono il prodotto desiderato.	***Cliente***	Cercare un supermercato tramite nome, catena o indirizzo di quest'ultimo.
Cliente	Visualizzare il catalogo del supermercato desiderato.	***Cliente***	Aggiungere un prodotto dal catalogo di un supermercato per poterlo acquistare.
Cliente	Rimuovere un prodotto precedentemente inserito nel carrello.	***Cliente***	Visualizzare il proprio carrello.
Cliente	Acquistare i prodotti nel proprio carrello.	***Cliente***	Registrarsi al sistema.
Referente	Rimuovere un prodotto dal catalogo del supermercato di afferenza.	***Referente***	Inserire un prodotto nel catalogo del supermercato di afferenza.

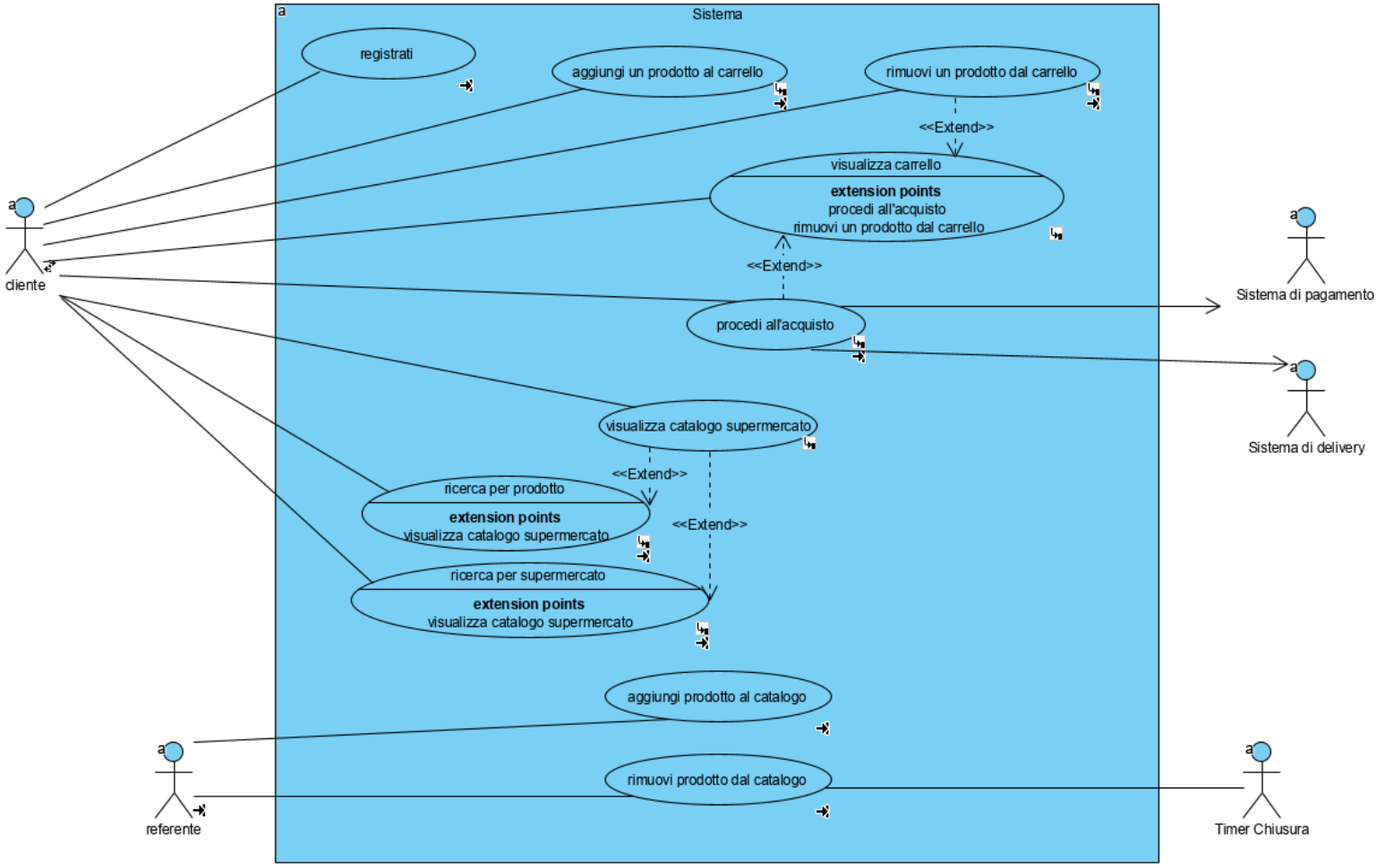

Figura 2.1: Use Case Diagram.

2.3 Glossario

A questo punto è utile stilare un glossario dei termini. Il glossario è utile in fase di inception per chiarire e ridurre le ambiguità che potrebbero scaturire dall'utilizzo dei termini specifici dell'applicazione. Nel glossario possono essere incluse le informazioni più varie, dagli attori alle relazioni tra essi.

Glossario

Termine	*Definizione ed informazioni*	*Formato*	*Regole di Validazione*	*Alias*
Prodotto	Un alimento prossimo alla scadenza.			Genere alimentare
Referente	Un dipendente del supermercato che interagisce con il software per il suo corretto funzionamento			Impiegato
Lista dei prodotti	È l'insieme dei prodotti disponibili attualmente presso un supermercato			Catalogo
Carrello	È una lista di prodotti, aggiornata dall'utente, pronti per l'acquisto.			
Servizio di delivery esterno	È un'entità esterna che mette a disposizione una flotta di fattorini disponibili a ritirare i prodotti dai supermercati ed a consegnarli agli utenti.			Servizio di consegna
Rider	Sono gli impiegati del servizio di delivery.			Fattorini

2.4 Diagramma di Contesto

Come artefatto finale della fase di inception si è scelto di realizzare un *context diagram*. Questo diagramma è utile non solo ai progettisti ma a tutti gli stakeholders, delimita i confini del sistema e ne mostra le interazioni ricevute e scaturite. Il software è modellato come una black-box, così come tutte le parti interagenti, e le associazioni descrivono in forma testuale lo scambio di informazioni tra questi.

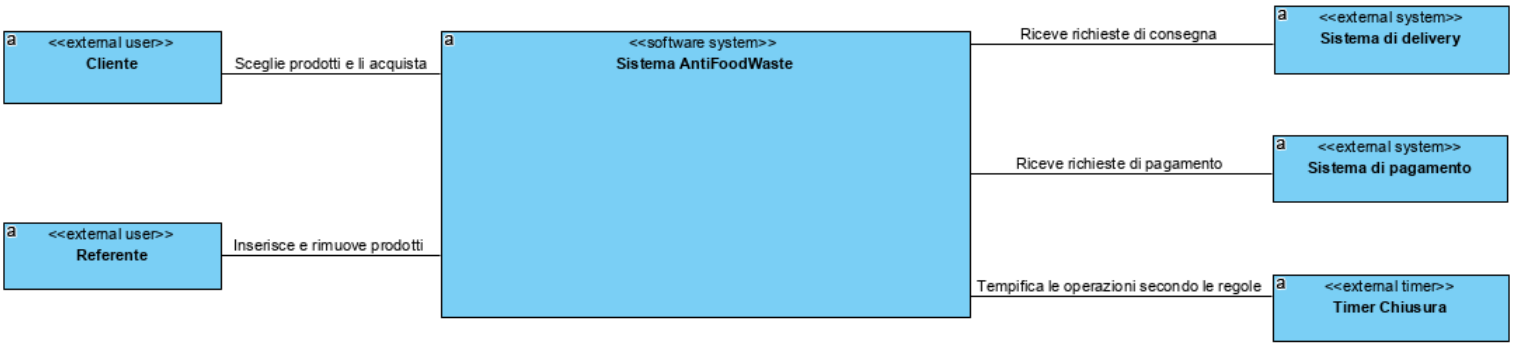

Figura 2.2: Context Diagram.

Capitolo 3

Prima Iterazione

Una volta conclusa la fase di ideazione si comincia, seguendo il modello di sviluppo UP, con la prima iterazione della fase di *elaborazione*. Di seguito è riportato lo schema generale dello sviluppo secondo UP. Come si evince dalla figura tutte le fasi (eccetto l'inception) sono divise in iterazioni.

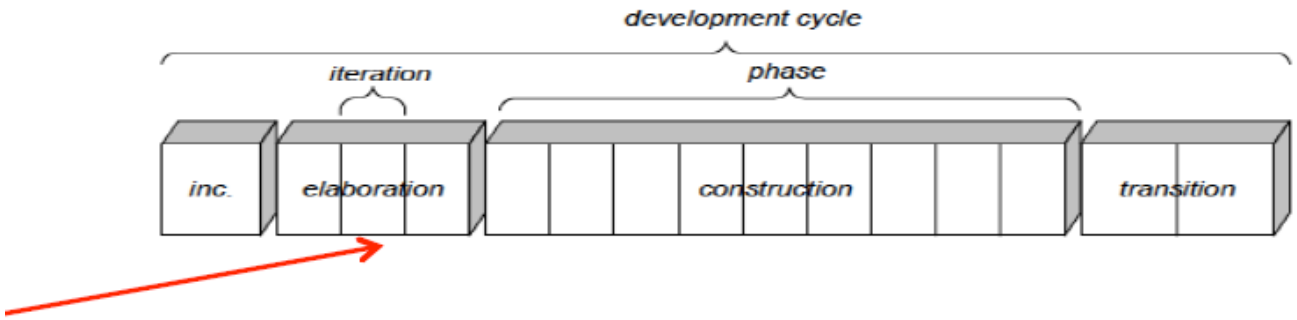

Figura 3.1: Ciclo di sviluppo UP.

3.1 Specifica Formale dei Casi d'Uso

Come prima cosa andiamo a dare una specifica formale completa dei casi d'uso che verranno implementati durante questa iterazione.

3.1.1 Ricerca per Supermercato

Caso d'uso:

Ricercare un supermercato per nome, catena o indirizzo.

Scope:

Sistema anti-foodwaste.

Primary Actor:

Utente.

Stakeholders and Interests:

- *Utente:* Vuole cercare un particolare supermercato per nome, catena o indirizzo. Vuole che la ricerca sia veloce e che non produca errori.
- *Proprietario Supermercato:* Vuole che i propri prodotti vengano messi in risalto e che sia semplice per gli utenti acquistarli e riceverli.

Preconditions:

L'utente è registrato ed autenticato.

Success Guarantee:

Viene mostrato un supermercato o una lista di supermercati che corrispondono in pieno o in parte ai termini della ricerca.

Main Success Scenario:

1. L'utente inserisce il testo della ricerca.
2. Inserisce il tipo di ricerca desiderata e clicca il bottone "RICERCA PER SUPERMERCATO".
3. Il sistema i supermercati che corrispondono alla ricerca.
4. Il sistema restituisce la lista di supermercati che corrispondono alla ricerca.

Extension:

3a. L'utente inserisce un testo di ricerca che non corrisponde ad alcun supermercato.

1. Il sistema restituisce un messaggio d'errore.

Special Requirements:

- Deve fornire un risultato velocemente.
- Deve fornire un' interfaccia utente semplice ed esteticamente piacevole.
- Deve essere robusto nel caso in cui il testo contenga caratteri non previsti.

Frequency of Occurrence:

Continuo

Open Issues:

3.1.2 Aggiungi al Carrello

Caso d'uso:

Aggiungere un prodotto al proprio carrello.

Scope:

Sistema anti food-waste

Level:

User-goal

Primary Actor:

Utente

Stakeholders and Interests:

- *Utente:* Vuole aggiungere al suo carrello temporaneo un determinato prodotto visualizzato a seguito di una ricerca. L'utente vuole avere un feedback visivo dell'esito dell'operazione. L'utente vuole ricevere tale feedback rapidamente.

Preconditions:

L'utente è registrato ed autenticato.

Success Guarantee:

Una unità di prodotto viene aggiunta al carrello temporaneo associato all'utente.

Main Success Scenario:

1. L'utente arriva sulla pagina del catalogo di un supermercato.
2. L'utente clicca il pulsante "Aggiungi al carrello" in corrispondenza del prodotto desiderato.
3. Il sistema aggiunge alla lista corrispondente al carrello il prodotto scelto.
4. Il sistema aggiorna la quantità disponibile di tale prodotto all'interno del catalogo del supermercato.

Extension:

2-3a. Il prodotto che si sta provando ad aggiungere al carrello non è più disponibile nel momento in cui lo si sta provando ad aggiungere al carrello

    1. Il sistema restituisce un messaggio d'errore ed annulla l'inserimento nel carrello.

Special Requirements:

- Il feedback visivo deve comunicare facilmente l'esito dell'operazione
- Deve essere semplice e rapido aggiungere diversi prodotti al carrello
- Il sistema deve garantire la consistenza dei dati, assicurando che il client ed il server lavorino sugli stessi dati

Frequency of Occurrence:

Potrebbe essere quasi continuo

Open Issues:

3.1.3 Acquisto

Caso d'uso:

Procedere all'acquisto dei prodotti nel carrello.

Scope:

Sistema anti-foodwaste.

Primary Actor:

Utente.

Stakeholders and Interests:

- *Utente:* Vuole acquistare i prodotti presenti nel proprio carrello affinchè gli vengano consegnati a casa nel minor tempo possibile. Vuole che il pagamento digitale sia sicuro e veloce, e che sia gestito da una terza parte affidabile e conosciuta.
- *Proprietario Supermercato:* Vuole che gli venga notificato l'ordine in maniera rapida e precisa, in modo da renderlo disponibile al servizio di consegna.
- *Servizio di consegna a domicilio:* Vuole ricevere le informazioni relative alla consegna velocemente e che queste siano le più accurate possibile.
- *Servizio esterno di pagamento:* Vuole ricevere le informazioni di pagamento in maniera sicura, secondo i protocolli più moderni.

Preconditions:

L'utente è registrato ed autenticato e ha un carrello non vuoto.

Success Guarantee:

L'ordine viene accettato, il pagamento registrato ed il servizio di delivery si attiva per la consegna.

Main Success Scenario:

1. L'utente clicca sull'icona del carrello.
2. L'utente si sposta nella pagina dove viene elencato il contenuto del carrello ed il relativo importo totale, a questo punto clicca sul tasto "ACQUISTA".
3. Il sistema produce allora la schermata di acquisizione dei dati di pagamento e l'utente inserisce i codici della carta di credito necessari.
4. I dati vengono validati, crittati ed inviati al sistema di pagamento esterno.

5. Viene ricevuta la notifica di pagamento andato a buon fine ed il sistema genera la ricevuta.

6. Il sistema notifica il gestore delle consegne inviando i dati del supermercato e del destinatario.

7. L'utente visualizza il messaggio di successo e le info relative.

Extension:

4a. I dati inseriti dall'utente non sono validi.

1. Viene mostrato un messaggio di errore e chiesto di riprovare.

5a. Il sistema di pagamento notifica la non riuscita della transazione.

1. Viene notificato il fallimento dell'operazione all'utente e la sessione di pagamento termina.

5b. Il sistema di pagamento non risponde in tempo e causa un **Timeout Error**

1. Viene notificato il timeout all'utente e la sessione di pagamento termina.

Special Requirements:

- Deve trattare i dati di pagamento nella maniera più sicura possibile
- Deve fornire un'interfaccia user-friendly
- Deve essere robusto e non creare inconsistenze

Frequency of Occurrence:

Frequente

Open Issues:

3.2 System Domain Model

Prima fase dell'elaborazione è la creazione del cosiddetto *system Domain Model*, chiamato anche *Modello degli Oggetti di Dominio* in ambito UP. Questo è il modello concettuale più importante dell'analisi OO, può essere fonte di ispirazione per la fase di progettazione e sarà input per diversi artefatti prodotti in seguito.

3.2.1 Modello di Dominio

Il modello di dominio è una visualizzazione degli oggetti di dominio, tipicamente realizzato a mezzo di un diagramma delle classi. Conterrà le classi concettuali, ottenute grazie all'analisi testuale del documento di specifica, i loro attributi e le relazioni tra esse.

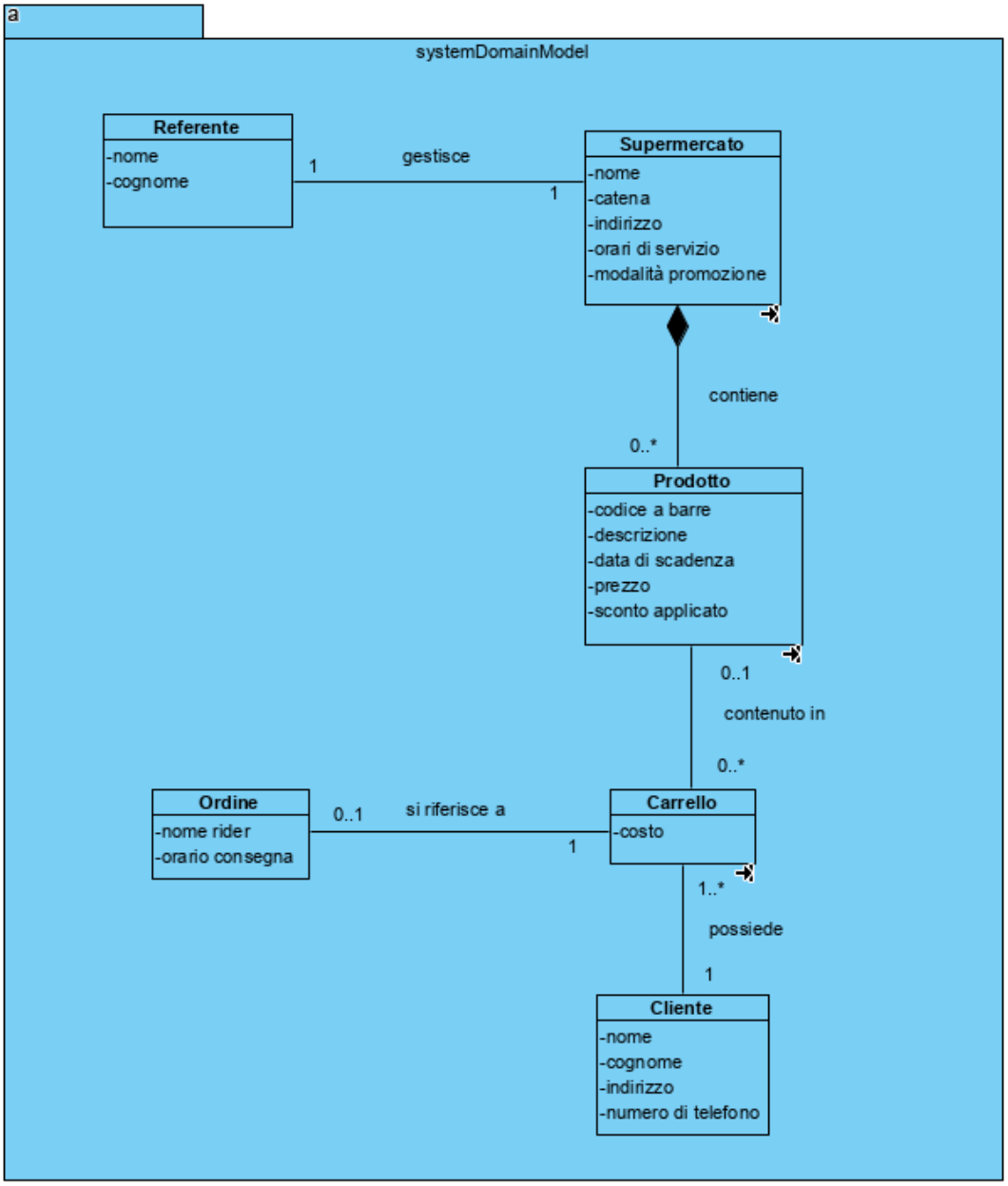

Figura 3.2: Modello di Dominio.

3.3 Verso la Progettazione

Per effettuare la transizione da fase di analisi a progettazione occorre identificare le operazioni svolte dal sistema e specificarne le interazioni richieste. Per fare ciò utilizziamo i cosiddetti *System Sequence Diagrams*. Un *SSD* è una vista del sistema che mostra, per un particolare scenario di caso d'uso, la sequenza di eventi generati dall'interazione dell'attore con il sistema. Per questo motivo la notazione utilizzata è quella tipica dei Sequence Diagrams. Di seguito verranno mostrati gli *SSD* realizzati.

3.3.1 SSD: Ricerca per Supermercato

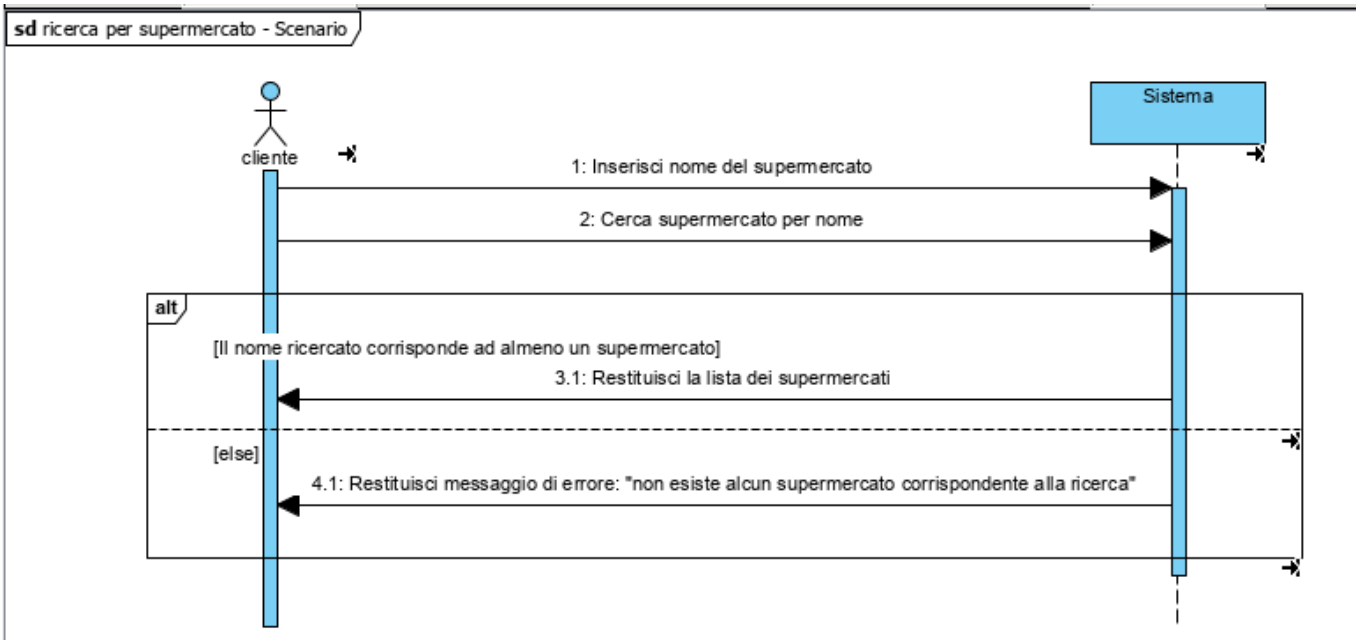

Figura 3.3: SSD: Ricerca per supermercato.

3.3.2 SSD: Aggiungi al Carrello

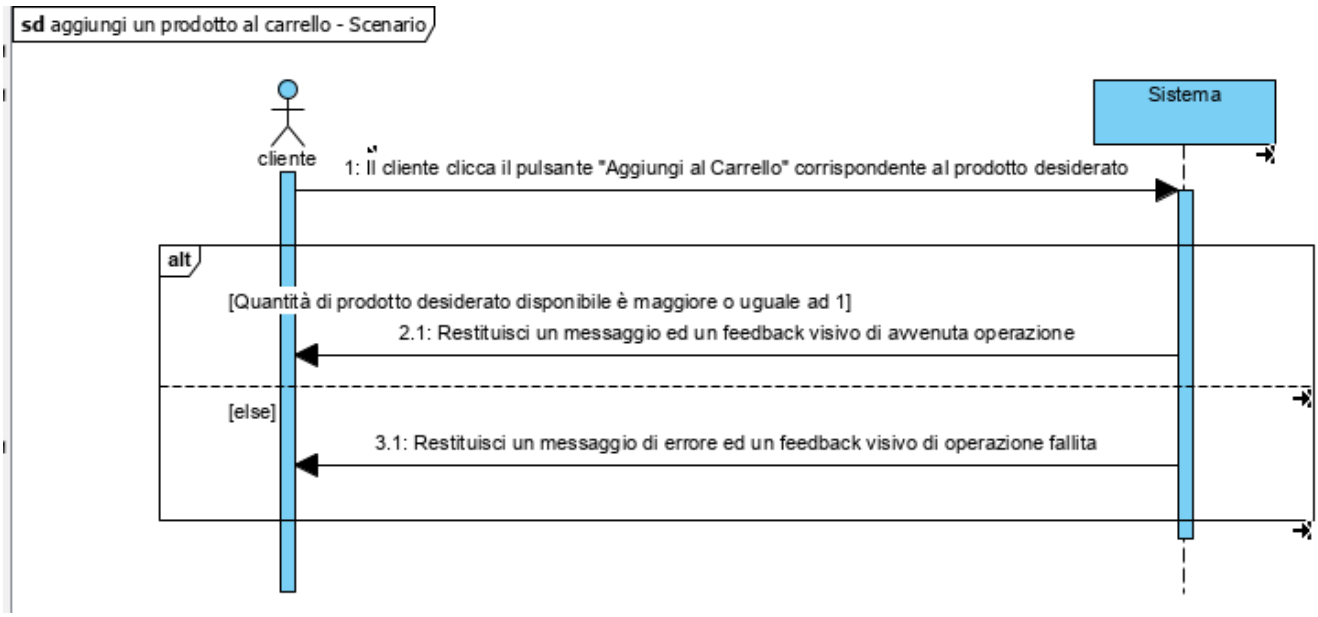

Figura 3.4: SSD: Aggiungi al Carrello.

3.3.3 SSD: Acquista

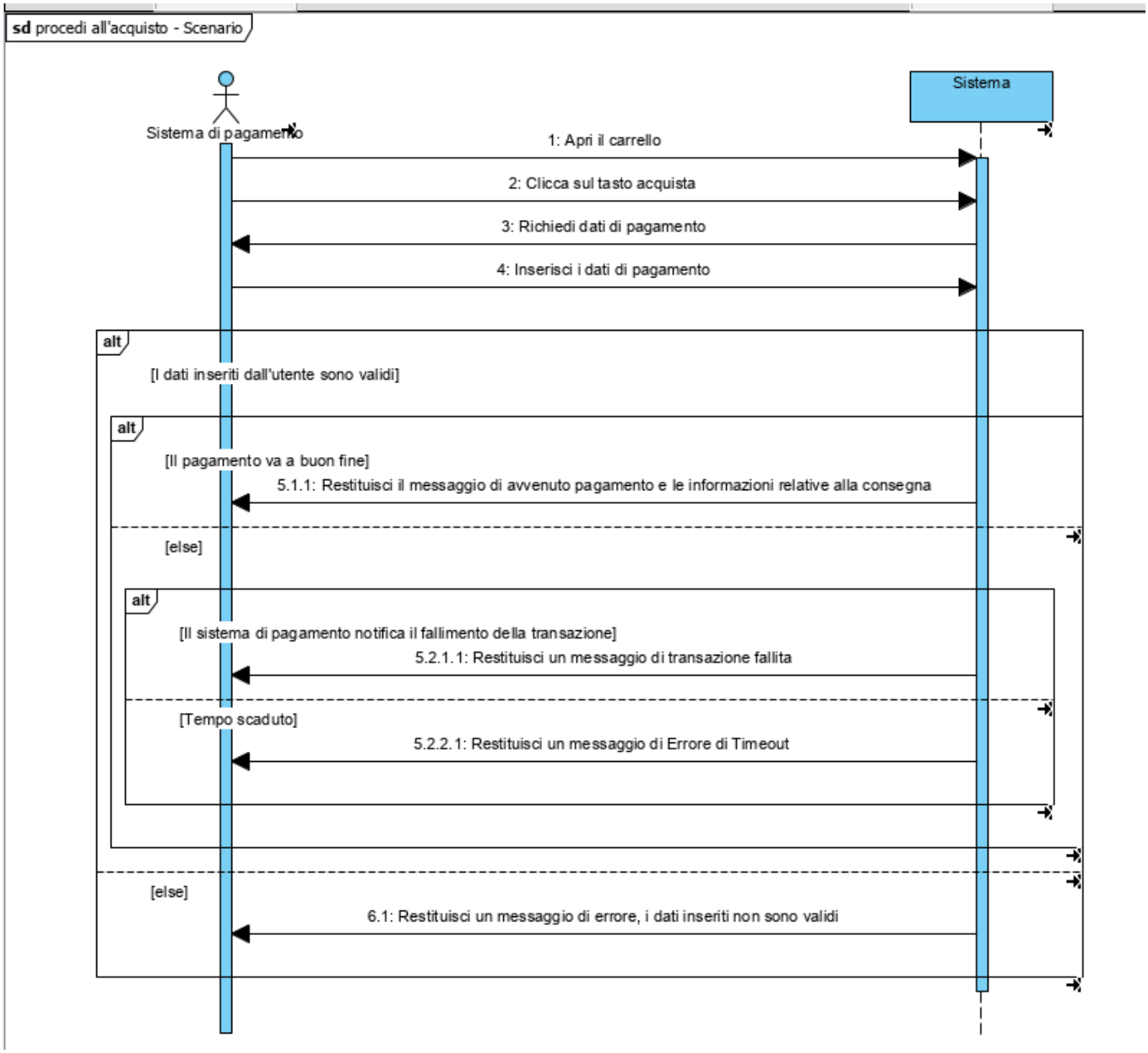

Figura 3.5: SSD: Acquista.

3.4 Architettura Logica

Arrivati a questo punto della prima iterazione è pratica comune cominciare la fase di progettazione tramite la definizione dell'architettura logica del sistema. Lo stile architetturale prescelto per la nostra applicazione è quello Client-Server; questo stile ben si adatta ai requisiti dato che è richiesta una centralizzazione dei dati e l'elaborazione può facilmente essere delegata ad una macchina remota, lasciando al dispositivo degli utenti solo un piccolo sottoinsieme di operazioni, principalmente relative alla visualizzazione dei risultati. All'interno di questi due macro-package troviamo le effettive suddivisioni dei livelli.

3.4.1 Package Diagram

Come si osserva dalla figura si è riusciti a mantenere una gestione *strict* dei livelli. Dal lato client il livello interfaccia utente si occuperà della visualizzazione e dell'acquisizione degli input. Il livello control farà da gestore andando a coordinare le operazioni, interfacciandosi sia con le entità di dominio che con in servizio di comunicazione remota che interagirà con il server. Dall'altro lato, invece, il server sarà in attesa di richieste, che verranno coordinate sempre a livello control. Le entità di dominio avranno poi compito di gestire la propria persistenza e di interagire con i servizi esterni.

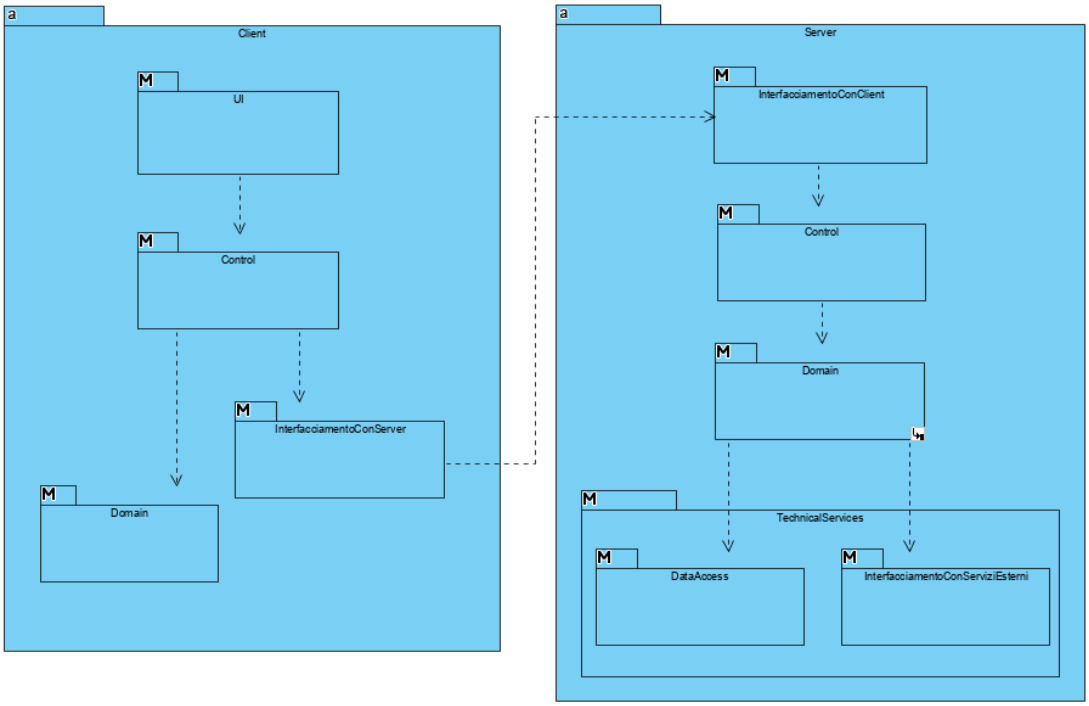

Figura 3.6: Package diagram dell'architettura.

3.5 Modellazione Statica

A partire dalla struttura del package diagram si procede con la modellazione statica delle classi del sistema e delle relazioni e interazioni tra di esse. La scelta delle classi di dominio è derivata dal SDM. A queste si aggiungono le classi utili a svolgere i compiti delineati nella sezione precedente. Primo di questo però, è stato realizzato un *subsystem class diagram*.

3.5.1 Subsystem Class Diagram

É stato realizzato un subsystem class diagram per evidenziare la relazione statica tra i sottosistemi che compongono l'applicazione. Questa vista è la più alta tra le viste statiche.

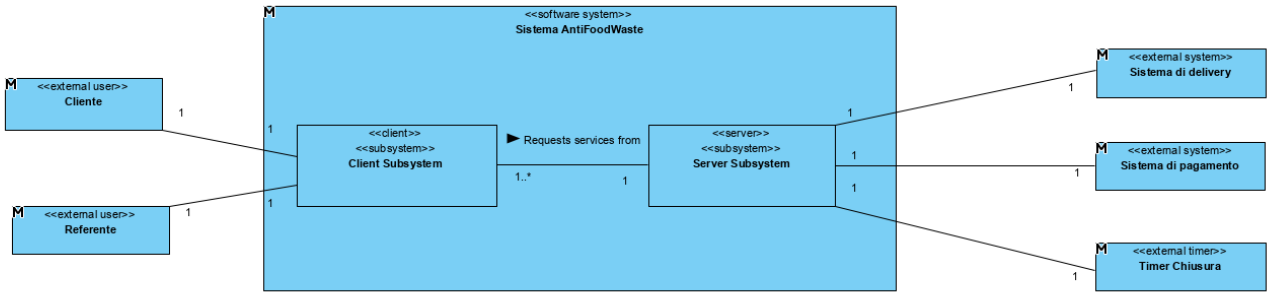

Figura 3.7: Subsystem Class Diagram - Client Side.

3.5.2 Class Diagram: Client

Lato Client troviamo le classi che si interfacciano con gli utenti, nel nostro caso una dedicata ai clienti ed un'altra dedicata agli addetti dei supermercati; queste classi andranno a utilizzare delle librerie basate su *swing* e *windowbuilder*. Una volta ottenuti i dati e delineate le richieste del cliente, lo strato UI invia le richieste al controller, che offre nella sua interfaccia tutti i metodi utili a portare avanti i casi d'uso. Il controller invia richieste al server tramite l'apposita classe proxy, che utilizza l'invocazione remota per comunicare con il server, e gestisce le entità di dominio.

3.5.3 Class Diagram: Server

Dal lato del server invece la classe di interfacciamento ascolta le richieste dei client e le inoltra al controller, il quale gestisce e delega le operazioni alle classi di dominio. Queste ultime hanno la responsabilità della propria persistenza, ed infatti

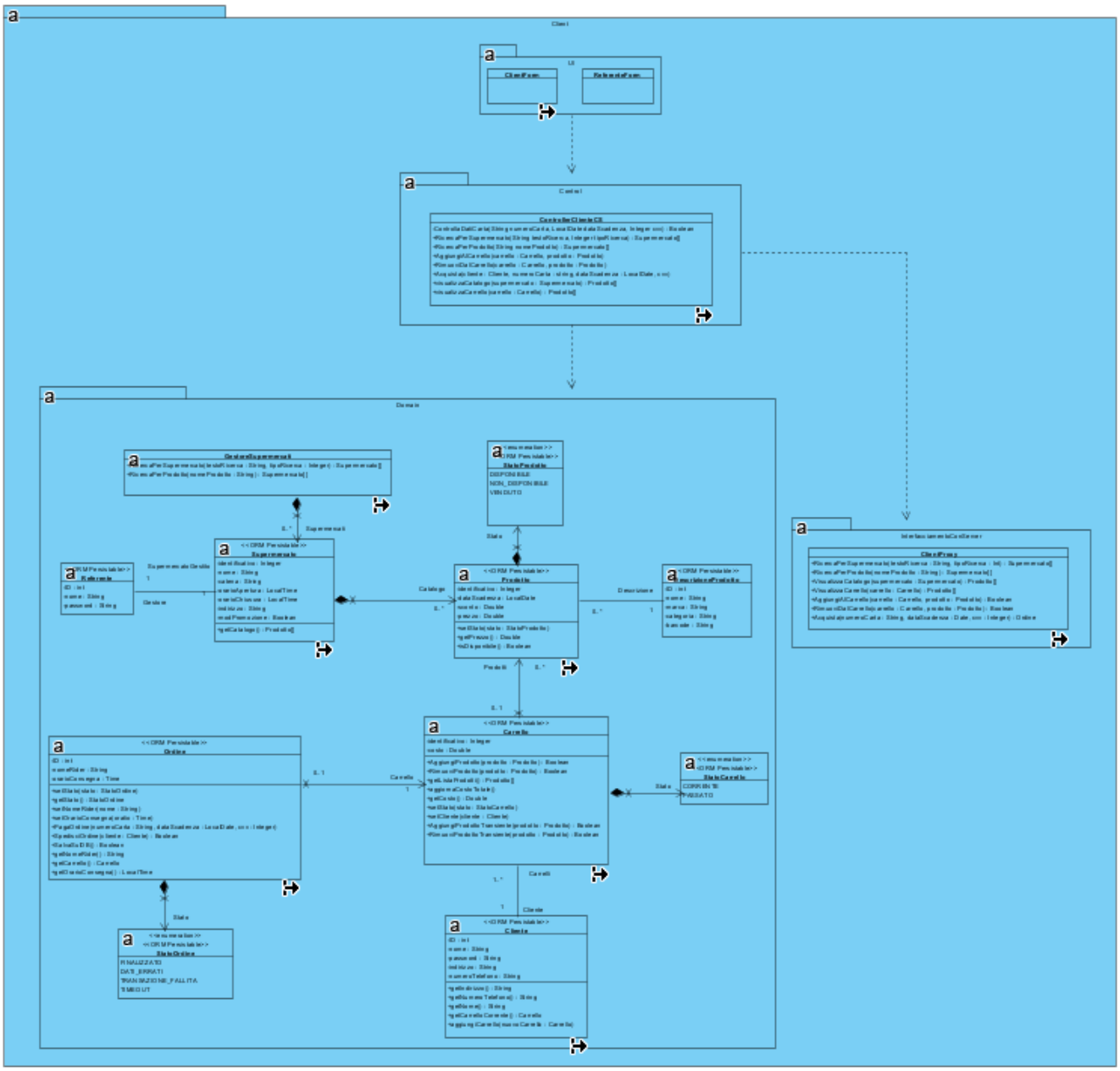

Figura 3.8: Design Class Diagram - Client Side.

comunicano direttamente con le classi database, ed inoltre, se necessario, gestiscono le richieste con i servizi esterni di pagamento e delivery. Alle entità che devono mantenere la persistenza è stato aggiunto lo stereotipo di «ORM persistable», utile per la traduzione automatica del modello OO in modello ER che vediamo in fig. 3.10.

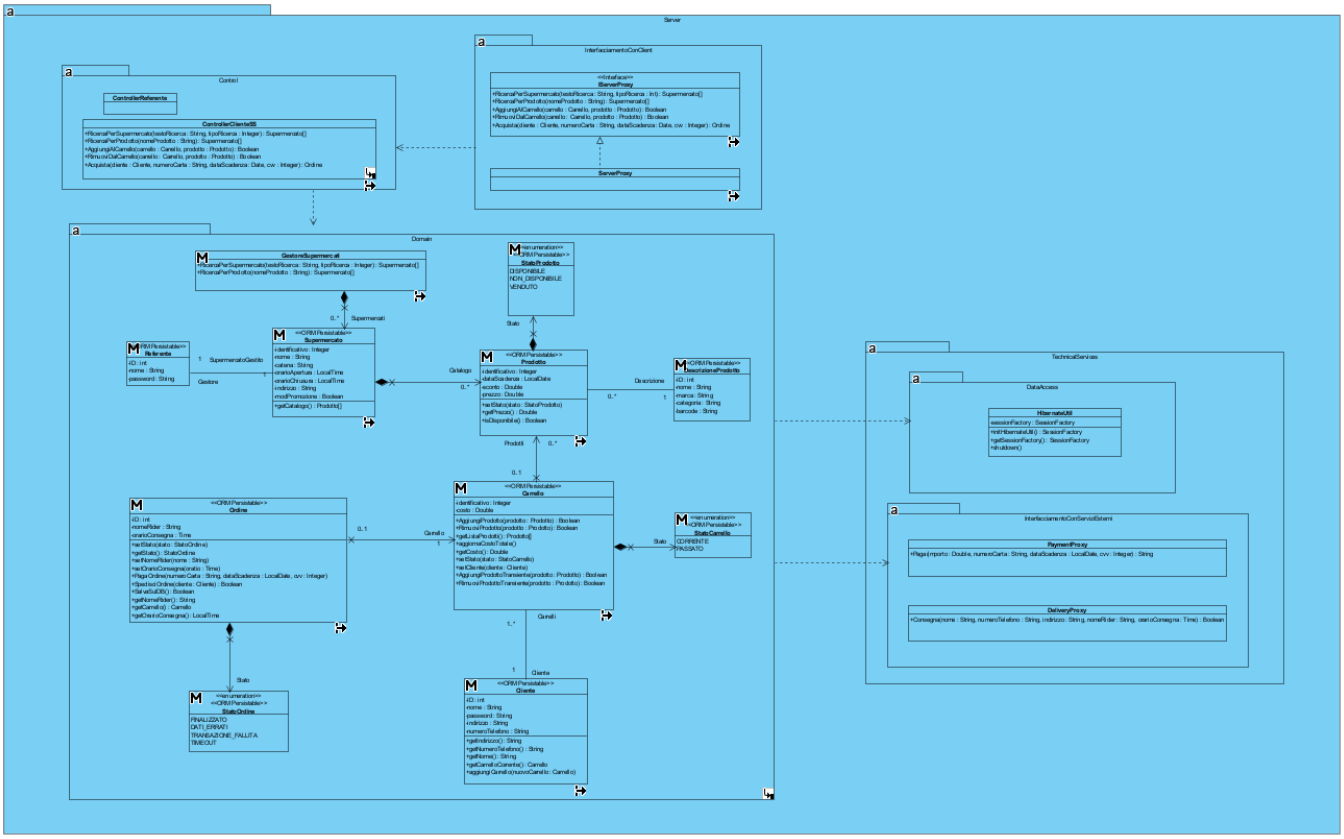

Figura 3.9: Design Class Diagram - Server Side.

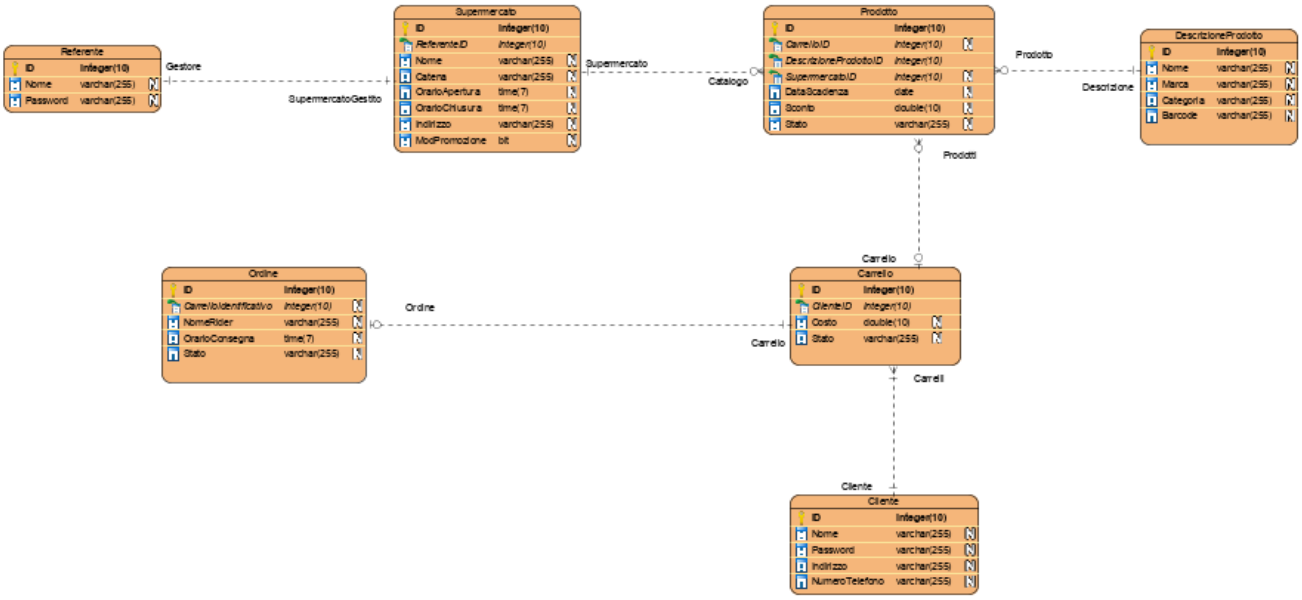

Figura 3.10: Diagramma Entità-Relazione Autogenerato.

3.6 Modellazione dinamica

Parallelamente alla modellazione statica si comincia anche a modellare le interazioni dinamiche del sistema, nel nostro caso a mezzo di Sequence Diagram. Di seguito sono riportati quelli degli scenari implementati nell'iterazione. É stata prestata particolare attenzione alla completezza dei sequence diagrams di progetto; grazie al loro dettaglio sarà infatti facile implementare le interazioni nel codice. Si è scelto di dividere in due parti quei diagrammi delle operazioni che interagiscono sia con il client che con il server per una questione di chiarezza e fruibilità.

3.6.1 Sequence Diagram: Ricerca per Supermercato

Client

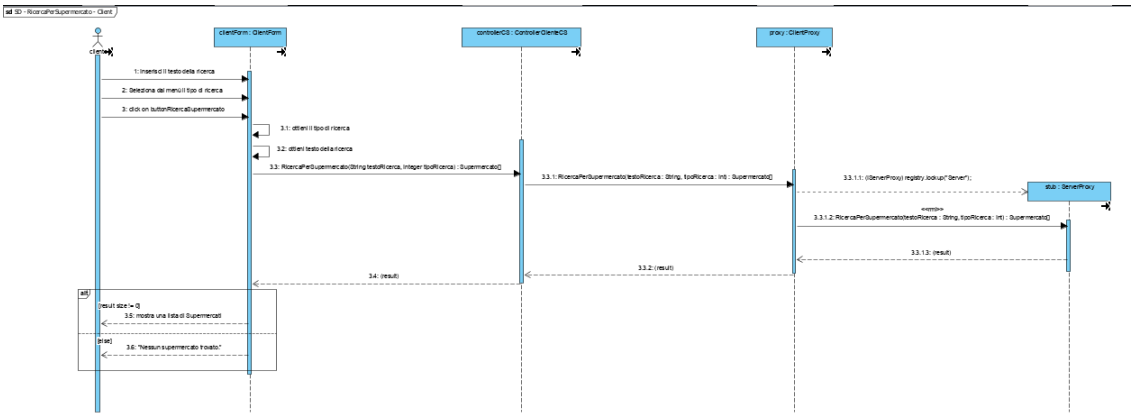

Figura 3.11: Sequence Diagram - Ricerca per Supermercato Client Side.

Server

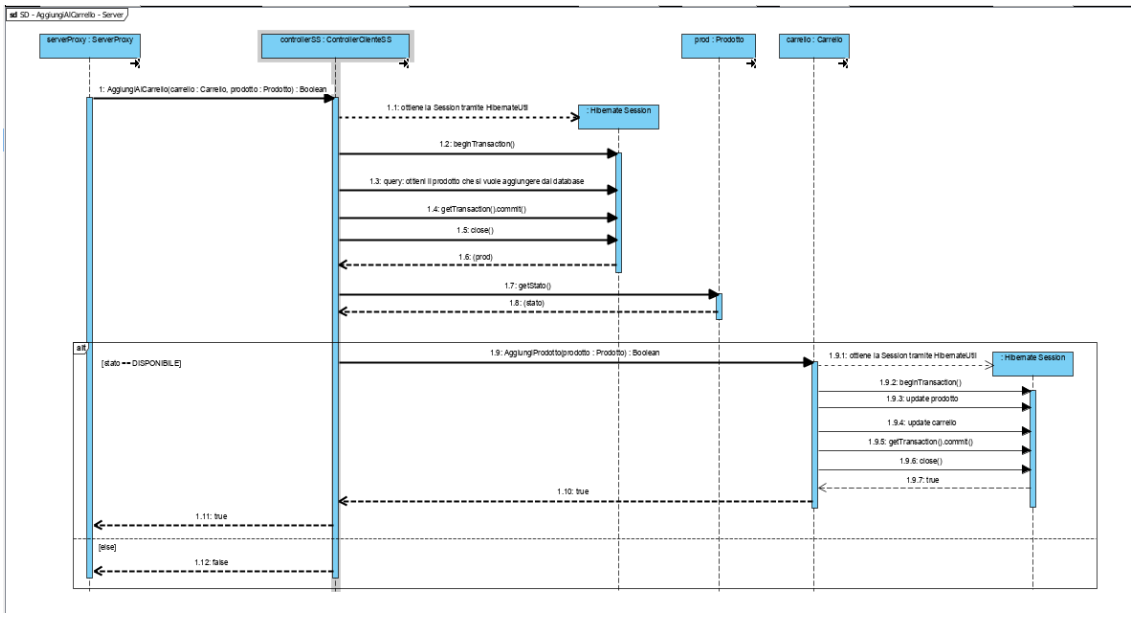

Figura 3.12: Sequence Diagram - Ricerca per Supermercato Server Side.

3.6.2 Sequence Diagram: Aggiungi al Carrello

Client

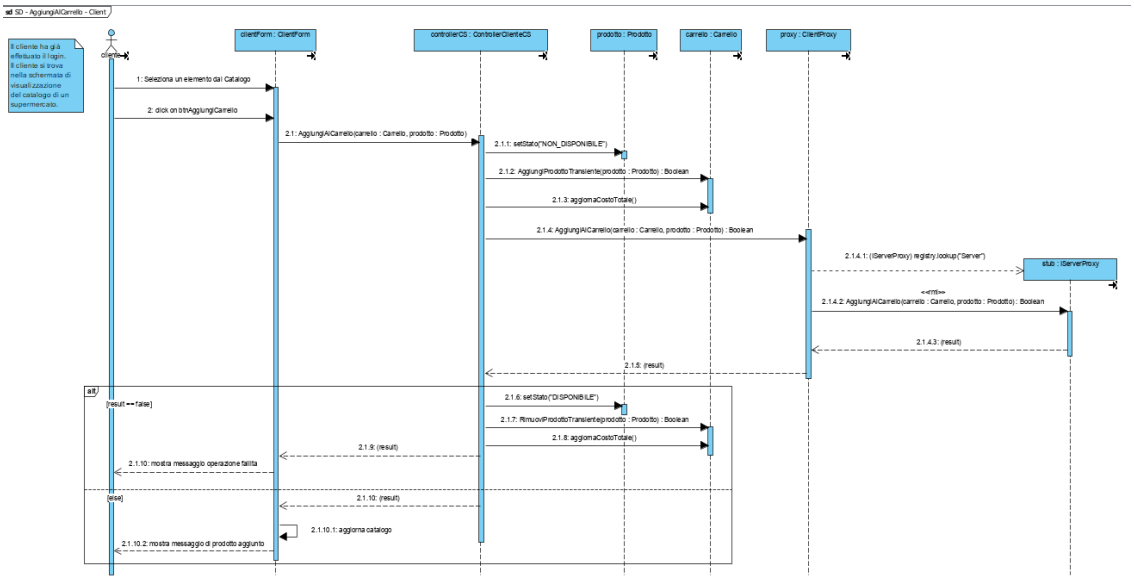

Figura 3.13: Sequence Diagram - Aggiungi al Carrello Client Side.

Sever

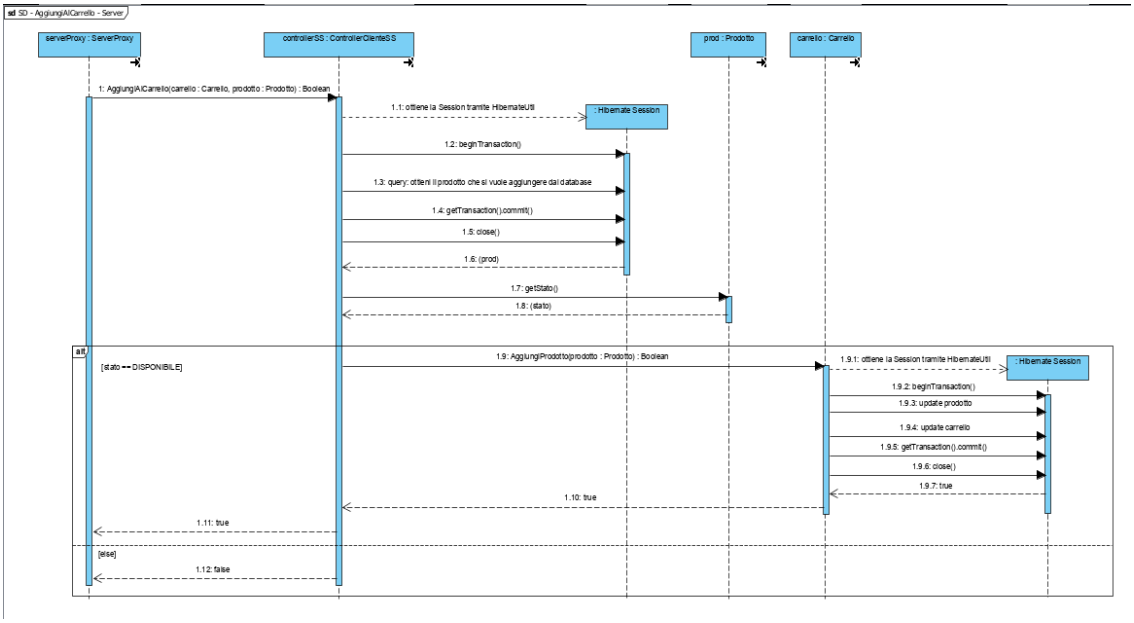

Figura 3.14: Sequence Diagram - Aggiungi al Carrello Server Side.

3.6.3 Sequence Diagram: Acquista

Client

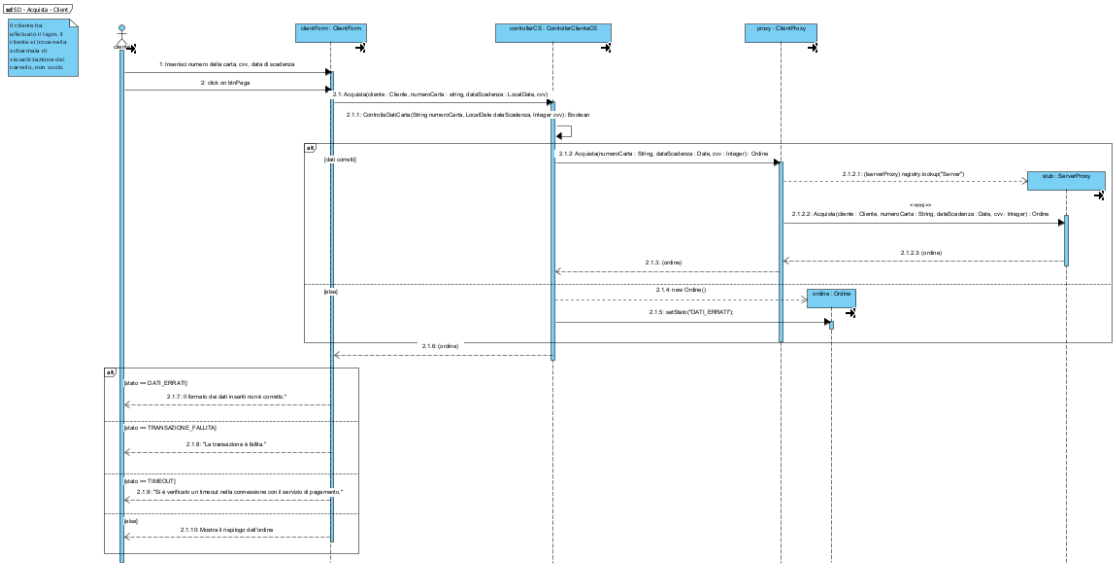

Figura 3.15: Sequence Diagram - Acquista Client Side.

Server

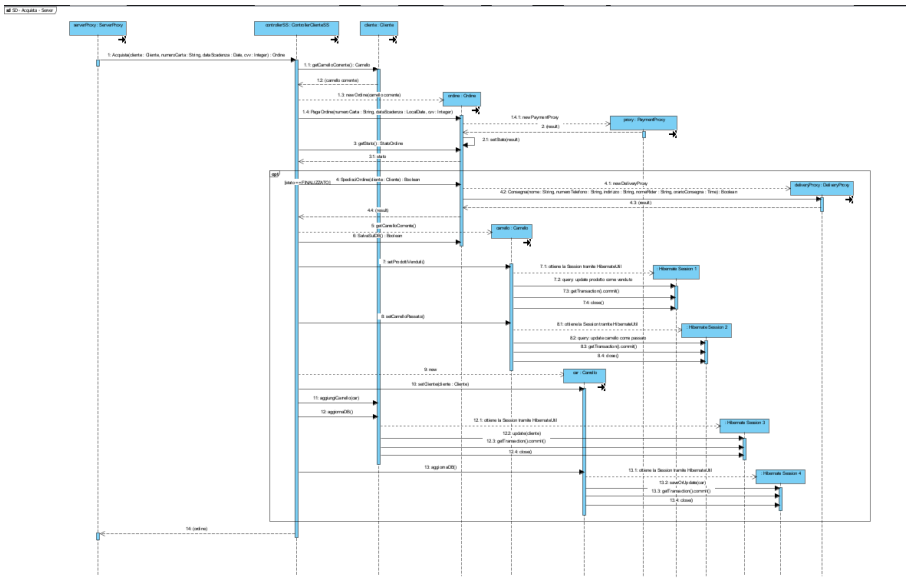

Figura 3.16: Sequence Diagram - Acquista Server Side.

3.7 Implementazione

Completata la stesura dei sequence si è passati alla scrittura del codice per queste operazioni. Innanzitutto è stato generato il codice a mezzo del tool preposto di VisualParadigm. É stato poi generato, a partire dal diagramma ER, il codice SQL per la creazione del DataBase. Di seguito si descriveranno strumenti, framework utilizzati e eventuali accorgimenti necessari al loro funzionamento.

3.7.1 Strumenti e Framework

Eclipse

L'IDE scelto per la costruzione del progetto è stato *Eclipse*, uno degli ambienti di sviluppo più completi ed utilizzati nel mondo Java. Questo permette la gestione semplificata dei file, l'esecuzione e il debugging locale. Il marketplace poi permette l'aggiunta di un'infinità di expansion packages che estendono le capacità del sistema.

Maven

Apache Maven è il tool di build automation prescelto per il sistema. Questo configura due aspetti fondamentali, la gestione delle dipendenze e la build. Tutto questo è descritto tramite un file xml configurabile dal programmatore secondo le proprie esigenze. Di seguito il contenuto del file.

pom.xml

Listing 3.1: pom.xml

```
1  <project xmlns="http://maven.apache.org/
     ↪ POM/4.0.0" xmlns:xsi="http://www.w3.
     ↪ org/2001/XMLSchema-instance"
     ↪ xsi:schemaLocation="http://maven.
     ↪ apache.org/POM/4.0.0 https://maven.
     ↪ apache.org/xsd/maven-4.0.0.xsd">
2  <modelVersion>4.0.0</modelVersion>
3  <groupId>PSSS</groupId>
4  <artifactId>progetto</artifactId>
5  <version>0.0.1-SNAPSHOT</version>
6  <name>Progetto di PSSS</name>
7  <dependencies>
8
9  <dependency>
10 <groupId>org.hibernate</groupId>
11 <artifactId>hibernate-core</artifactId>
12 <version>5.4.18.Final</version>
13 </dependency>
14
15 <dependency>
16 <groupId>mysql</groupId>
17 <artifactId>mysql-connector-java</
     ↪ artifactId>
18 <version>8.0.21</version>
19 </dependency>
20
21 </dependencies>
22 </project>
```

Hibernate

Per la gestione e la comunicazione con il DB è stato scelto *Hibernate*, middleware Open-Source che, attraverso il proprio framework, offre un servizio di Object-Relational Mapping. Hibernate si pone a metà tra il livello della business logic dell'applicazione e il RDBMS, offrendo tutti i servizi CRUD in una veste semplificata per lo sviluppatore.

RMI

Per la comunicazione tra i sistemi, sia tra gli stessi client e server, che tra il server ed i servizi esterni, è stata utilizzata l'API Java *RMI*. RMI, o Remote Method Invocation,

ha come scopo quello di rendere invisibile al programmatore la maggioranza dei dettagli della comunicazione su rete. La comunicazione avviene tramite degli *stub*, che comunicano attraverso delle interfacce predisposte dal fornitore di servizi. Il fornitore si registra sul *registry*, specificando il porto su cui vuole comunicare (quella di default è la 1099), e chi voglia accedere ai servizi forniti fa il lookup sul registry e invia richieste secondo l'interfaccia.

WindowBuilder

Windowbuilder è un tool di creazione per interfacce grafiche in Java. Esso è composto sia da SWT Designer che Swing Designer ed adotta una filosofia WYSIWYG che rende molto intuitiva il design. Ha un approccio event-driven che permette di specificare semplicemente le risposte ad un'azione, attraverso una serie di *listeners*.

3.7.2 Codice

La struttura del codice segue quella del package diagram. Per visualizzare il codice si rimanda alla cartella del progetto.

Capitolo 4

Seconda Iterazione

Nella seconda iterazione andiamo a formalizzare i rimanenti UC, arrivando fino ai sequence diagram di progetto, per poi procedere alla codifica, il testing ed il deployment dell'applicazione

4.1 Specifica Formale dei Casi d'Uso

Come prima cosa andiamo a dare una specifica formale completa dei casi d'uso che verranno implementati durante questa iterazione.

4.1.1 Ricerca per Prodotto

Caso d'uso:

Ricercare un prodotto tra i supermercati.

Scope:

Sistema anti-foodwaste.

Primary Actor:

Utente.

Stakeholders and Interests:

- *Utente:* Vuole cercare un particolare prodotto e ricevere la lista dei supermercati che ne dispongono. Vuole che la ricerca sia veloce e che non produca errori.
- *Proprietario Supermercato:* Vuole che i propri prodotti vengano messi in risalto e che sia semplice per gli utenti acquistarli e riceverli.

Preconditions:

L'utente è registrato ed autenticato.

Success Guarantee:

Viene mostrata una lista di supermercati nei quali è disponibile almeno un'unità del prodotto cercato, corredata da una breve lista di informazioni utili.

Main Success Scenario:

1. L'utente clicca sul pulsante "RICERCA PER PRODOTTO".
2. Inserisce il prodotto desiderato e clicca il tasto "INVIA".
3. Il sistema ricerca il prodotto dal catalogo di ciascun supermercato disponibile.
4. Il sistema restituisce una lista di supermercati che dispongono del prodotto cercato.

Extension:

3a. L'utente inserisce un prodotto non disponibile in nessun supermercato.

1. Il sistema restituisce un messaggio d'errore.

Special Requirements:

- Deve fornire un risultato velocemente.
- Deve fornire un' interfaccia utente semplice ed esteticamente piacevole.
- Deve essere robusto nel caso in cui il testo contenga caratteri non previsti.

Frequency of Occurrence:

Continuo

Open Issues:

4.1.2 Visualizza Catalogo

Caso d'uso:

Visualizza il catalogo di un supermercato.

Scope:

Sistema anti-foodwaste

Primary Actor:

Utente

Stakeholders and Interests:

- *Utente:* Vuole avere una visione chiara dell'intero catalogo del supermercato. La lista deve essere ordinata in ordine alfabetico.
- *Proprietario Supermercato:* Vuole che l'utente sia invogliato ad acquistare i suoi prodotti, in modo da massimizzare le vendite e minimizzare gli sprechi.

Preconditions:

L'utente è registrato e autenticato.

L'utente deve aver individuato il supermercato in seguito ad una ricerca.

Success Guarantee:

L'utente visualizza l'intero catalogo di prodotti messi a disposizione dal supermercato.

Main Success Scenario:

1. L'utente individua un supermercato dalla lista ottenuta da una ricerca.
2. L'utente clicca il pulsante 'VISUALIZZA CATALOGO'.
3. L'utente visualizza una lista dei prodotti offerti dal supermercato individuato.

Extension:

3a. Il supermercato individuato ha terminato l'orario di servizio.

1. Il sistema restituisce un avviso a riguardo della chiusura del supermercato, permette di visualizzarne il catalogo ma non permette di aggiungere elementi al carrello.

Special Requirements:

- Il catalogo mostrato deve essere aggiornato al momento della visita.
- Lo scorrimento dei prodotti deve essere fluido ed intuitivo.

Frequency of Occurrence:

Potrebbe essere quasi continuo.

Open Issues:

4.1.3 Visualizza Carrello

Caso d'uso:

Visualizza il carrello.

Scope:

Sistema anti-foodwaste

Primary Actor:

Utente

Stakeholders and Interests:

- *Utente:* Vuole visualizzare il contenuto corrente del proprio carrello ed il costo totale per l'acquisto.

Preconditions:

L'utente è registrato e autenticato.

Success Guarantee:

L'utente visualizza il contenuto del proprio carrello ed il totale correlato senza errori.

Main Success Scenario:

1. L'utente clicca sul bottone 'VISUALIZZA CARRELLO'.
2. Il sistema mostra la lista aggiornata del carrello del cliente.

Extension:

- 1.

Special Requirements:

-
-
-

Frequency of Occurrence:

Open Issues:

4.1.4 Rimuovi dal Carrello

Caso d'uso:

Rimuovere un prodotto dal carrello.

Scope:

Sistema anti-foodwaste

Primary Actor:

Utente

Stakeholders and Interests:

- *Utente:* Vuole facilmente rimuovere un'unità del prodotto specificato dal suo attuale carrello.
- *Proprietario Supermercato:* Vuole che i clienti effettuino un ordine contente solo gli articoli effettivamente desiderati e nella quantità desiderata.
- *Rider:* Non vuole trasportare più prodotti di quelli realmente desiderati dal cliente, in modo da massimizzare i prodotti trasportati.

Preconditions:

L'utente è registrato e autenticato.

Nel carrello è presente almeno un'unità del prodotto specificato.

Success Guarantee:

Se nel carrello era presente un'unica unità del prodotto specificato, il carrello contiene tutti i prodotti presenti precedentemente tranne quello specificato. In caso contrario contiene tutti i prodotti presenti precedentemente, ma il prodotto specificato presenta un'unità in meno.

Main Success Scenario:

1. L'utente clicca sul pulsante "CARRELLO".
2. L'utente seleziona dalla lista di prodotti presenti nel carrello un prodotto di cui desidera rimuovere un'unità.
3. Il sistema rimuove un'unità del prodotto specificato dal carrello.
4. Il sistema aggiorna la quantità disponibile del prodotto specificato all'interno del catalogo del supermercato corrispondente.

Extension:

2a. Il carrello è vuoto.

1. L'utente non può rimuovere alcun elemento dal carrello.

Special Requirements:

- Deve rimuovere velocemente un'unità del prodotto specificato dal carrello.
- Dev'essere robusto nel caso in cui l'utente provi a rimuovere un'unità del prodotto prima che il carrello si sia aggiornato in seguito alla precedente operazione di rimozione.

Frequency of Occurrence:

Frequente.

Open Issues:

4.2 Verso la progettazione

Per effettuare la transizione da fase di analisi a progettazione occorre identificare le operazioni svolte dal sistema e specificarne le interazioni richieste. Per fare ciò utilizziamo i cosiddetti *System Sequence Diagrams*. Un *SSD* è una vista del sistema che mostra, per un particolare scenario di caso d'uso, la sequenza di eventi generati dall'interazione dell'attore con il sistema. Per questo motivo la notazione utilizzata è quella tipica dei Sequence Diagrams. Di seguito verranno mostrati gli *SSD* realizzati.

4.2.1 SSD: Ricerca per Prodotto

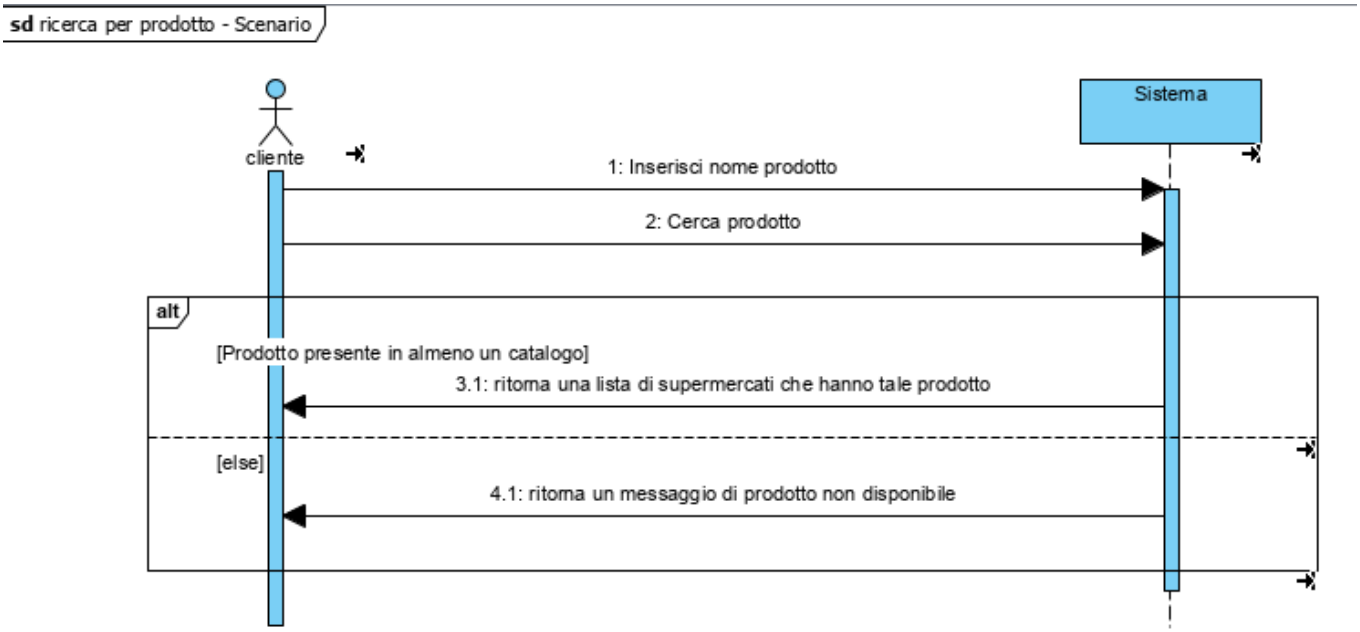

Figura 4.1: SSD: Ricerca per Prodotto.

4.2.2 SSD: Visualizza Catalogo

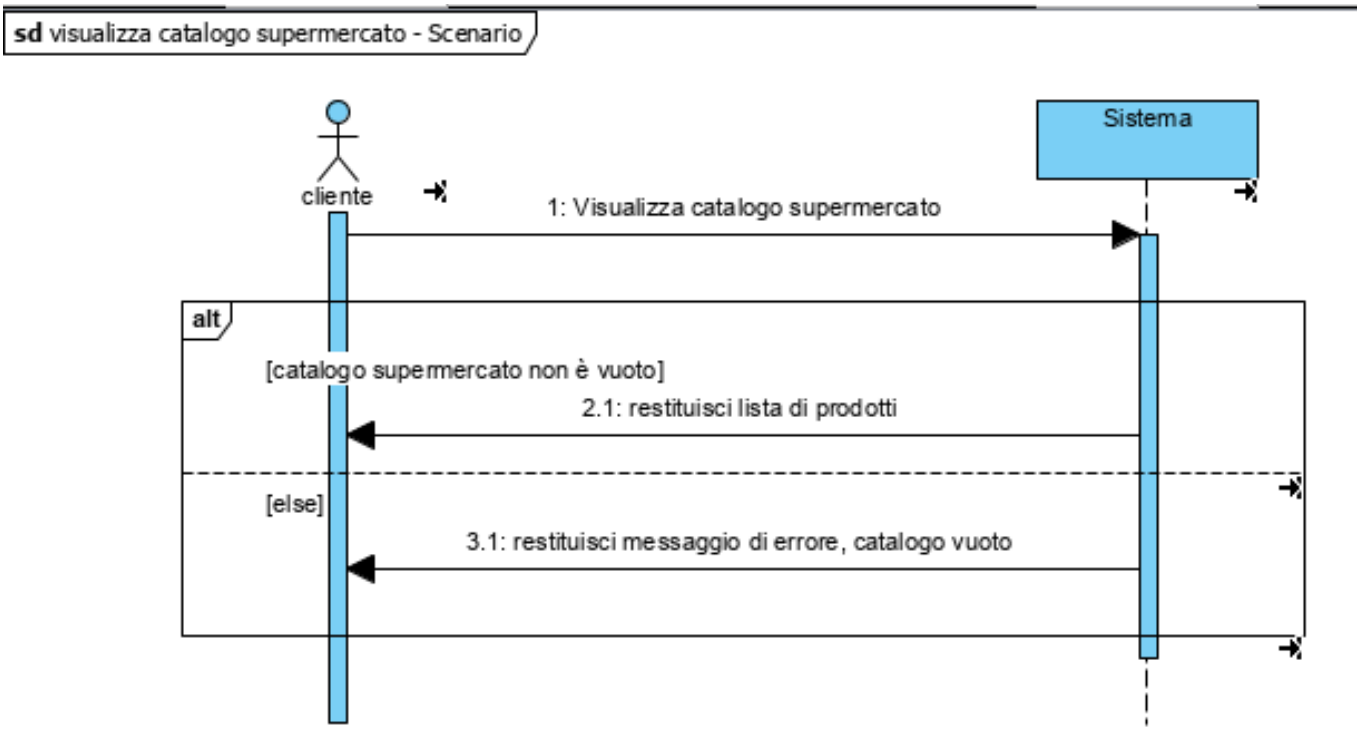

Figura 4.2: SSD: Visualizza Catalogo.

4.2.3 SSD: Visualizza Carrello

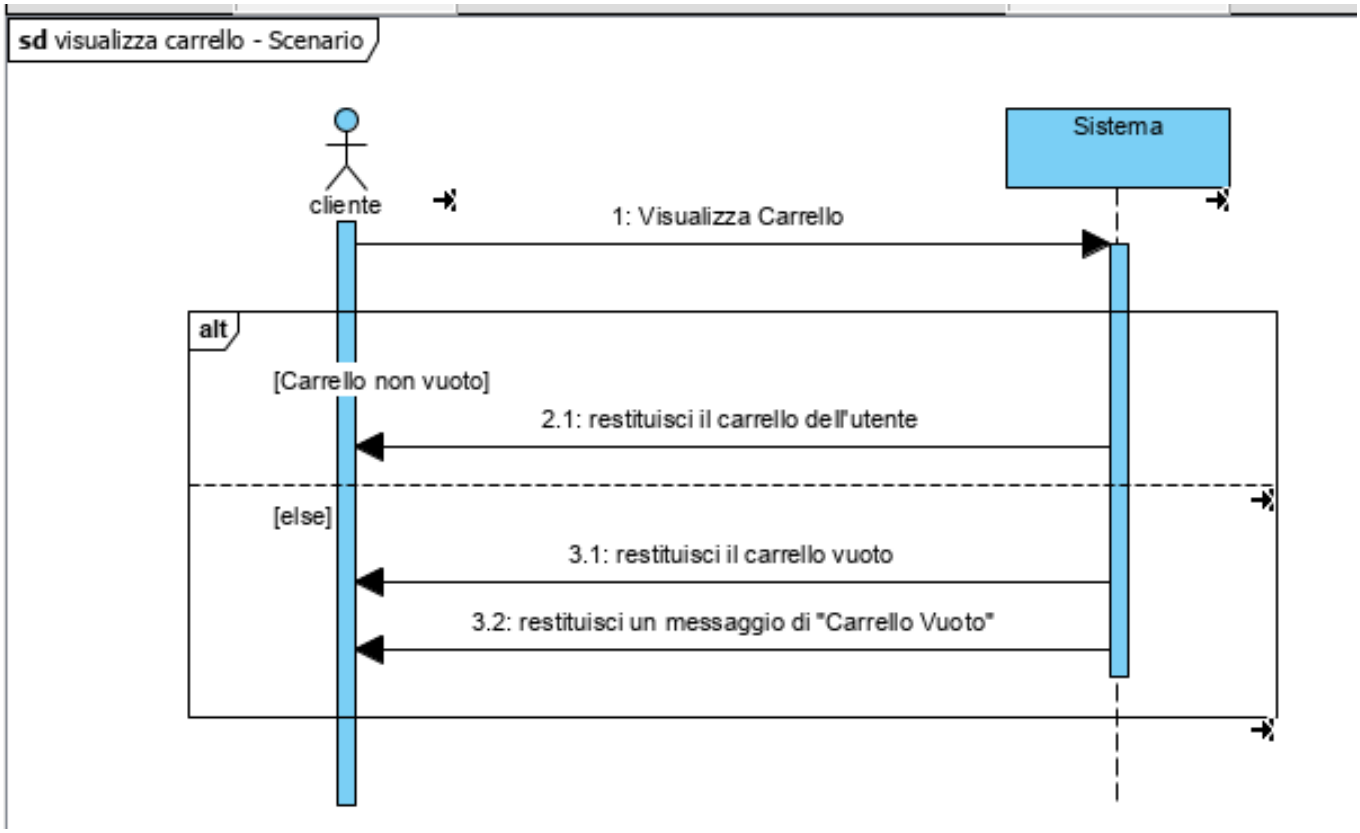

Figura 4.3: SSD: Visualizza Carrello.

4.2.4 SSD: Rimuovi dal Carrello

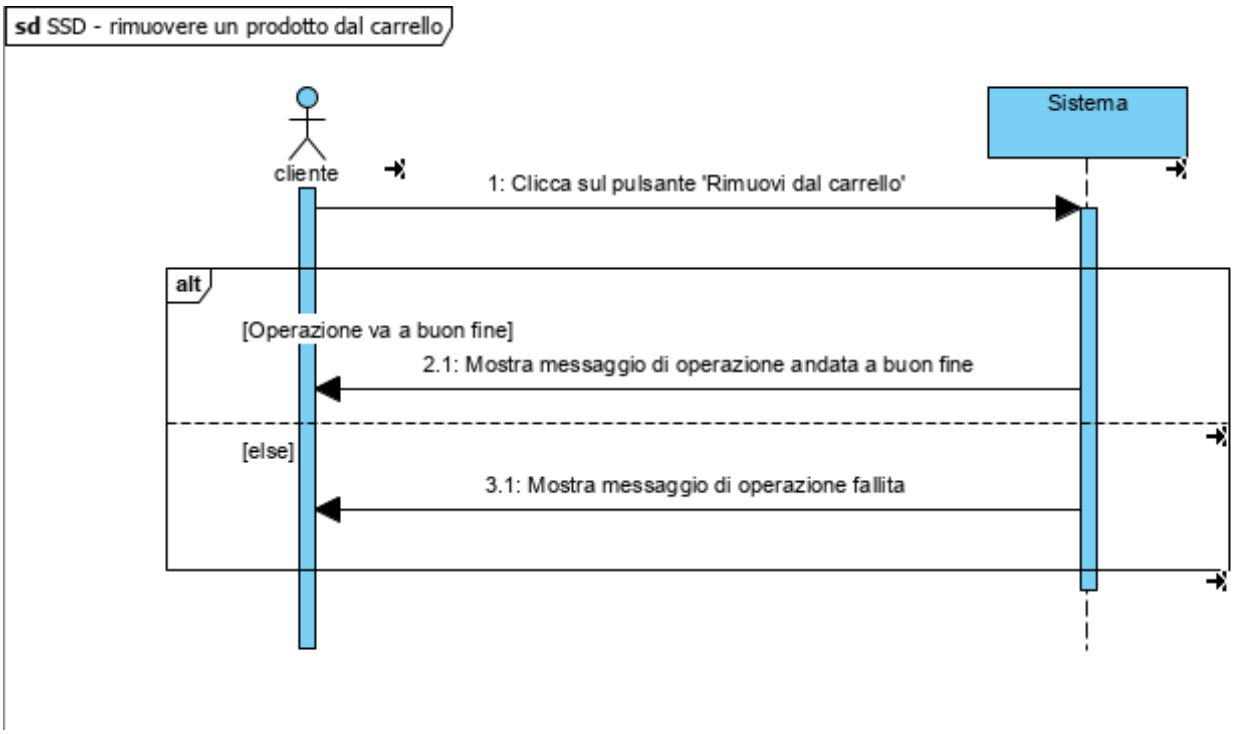

Figura 4.4: SSD: Rimuovi dal Carrello.

4.3 Modellazione dinamica

4.3.1 Sequence Diagram: Ricerca per Prodotto

Client

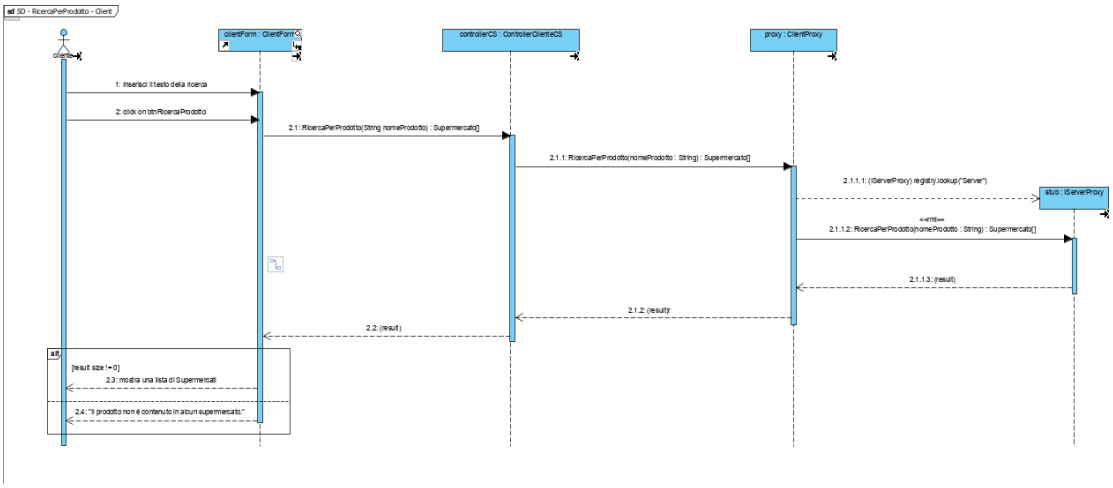

Figura 4.5: Sequence Diagram - Ricerca per Prodotto Client Side.

Server

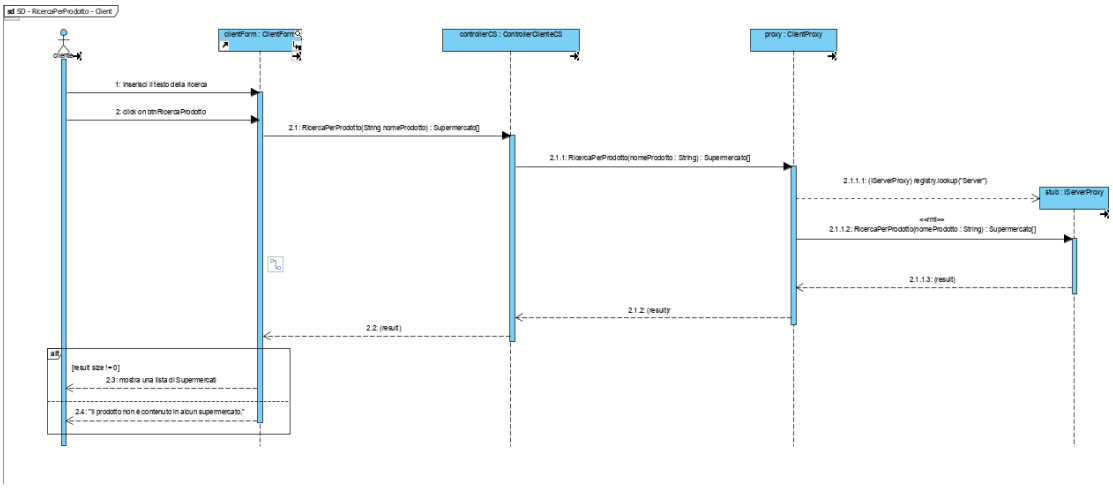

Figura 4.6: Sequence Diagram - Ricerca per Prodotto Client Side.

4.3.2 Sequence Diagram: Visualizza Catalogo

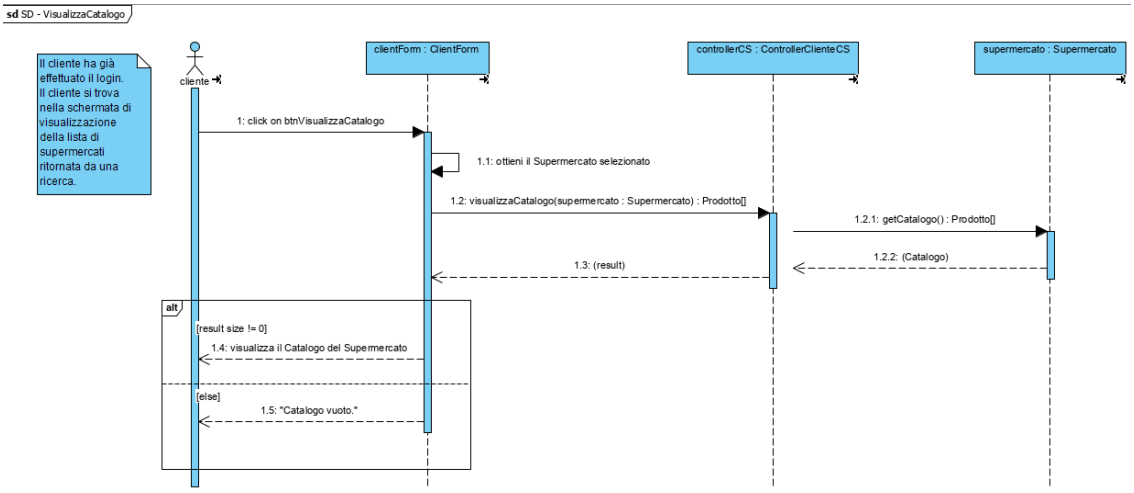

Figura 4.7: Sequence Diagram - Visualizza Catalogo.

4.3.3 Sequence Diagram: Visualizza Carrello

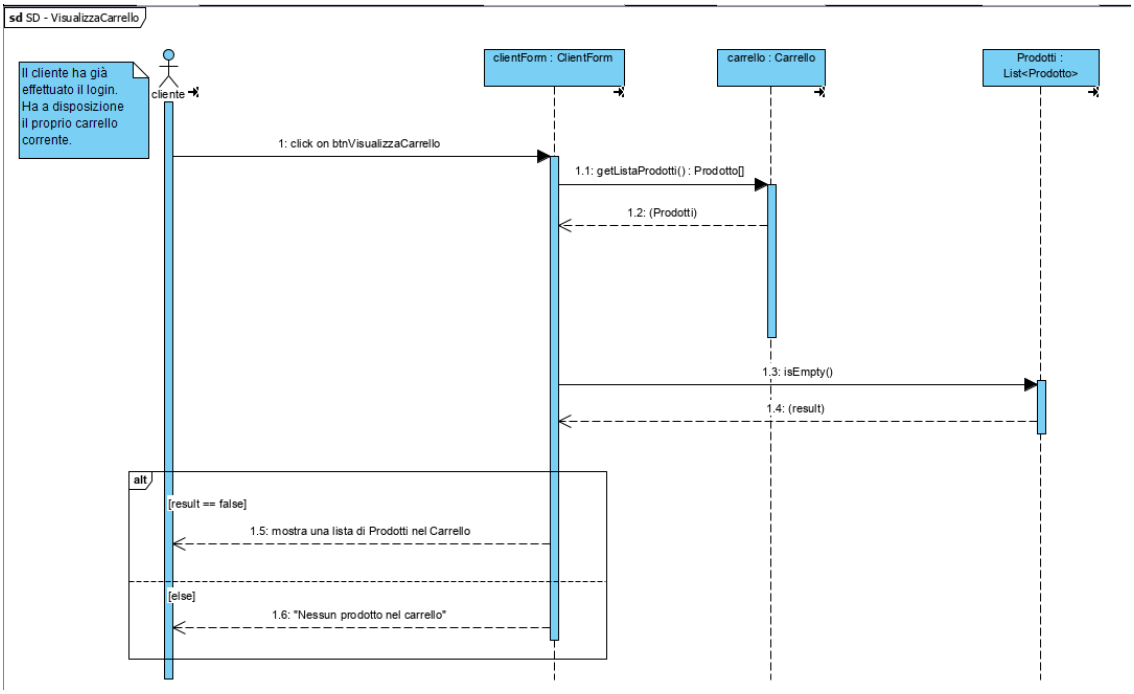

Figura 4.8: Sequence Diagram - Visualizza Carrello.

4.3.4 Sequence Diagram: Rimuovi dal Carrello

Client

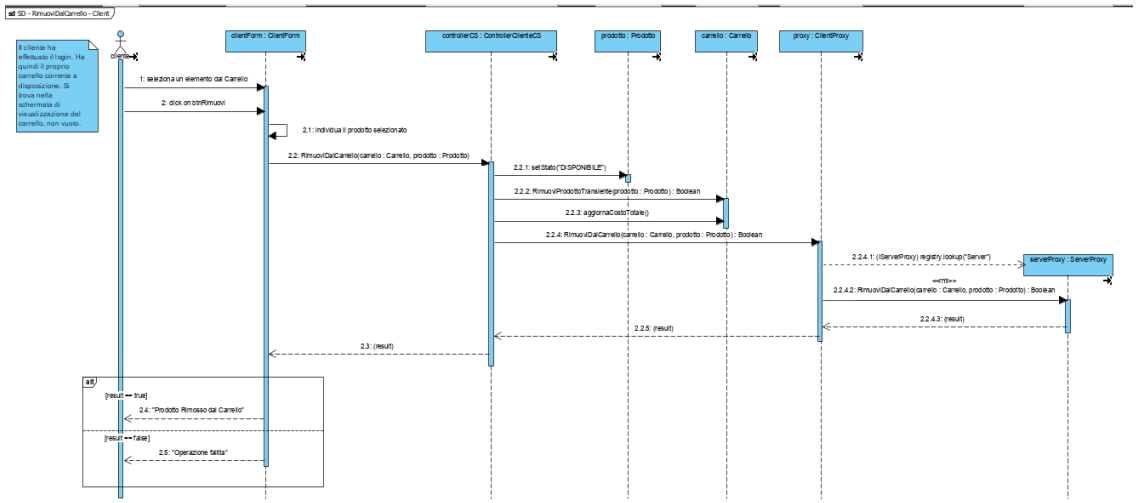

Figura 4.9: Sequence Diagram - Rimuovi dal Carrello Client Side.

Server

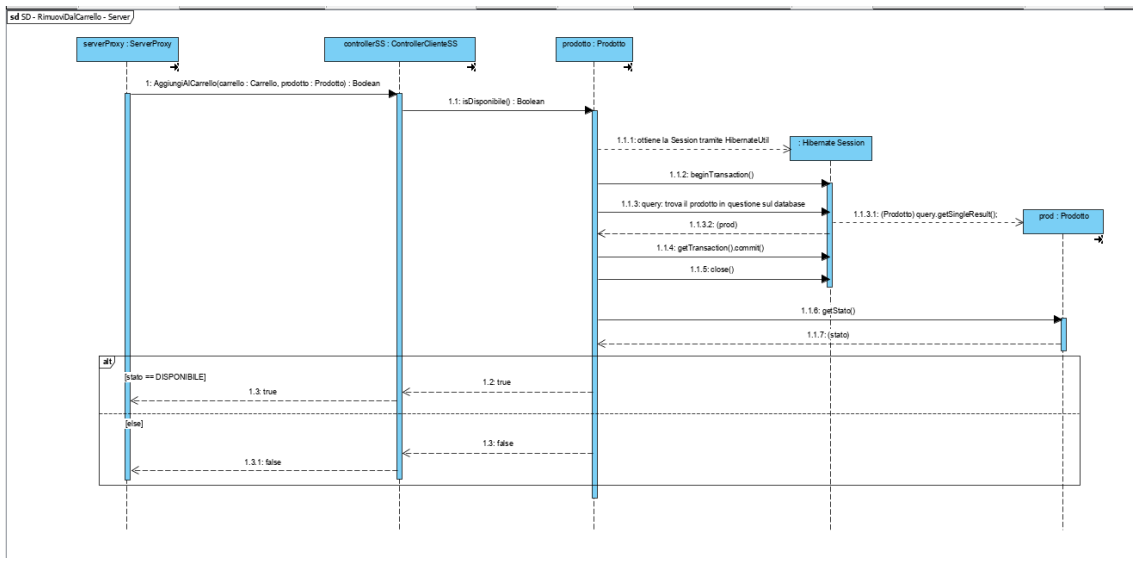

Figura 4.10: Sequence Diagram - Rimuovi dal Carrello Server Side.

4.4 Testing

L'applicazione è stata testata per valutarne la robustezza. Casi di test verificati sono stati ad esempio:

- Interazione contemporanea di vari client con il server.
- Acquisto di prodotti in stati non validi.
- Inserimento di credenziali di pagamento invalide.
- Acquisto di carrello in stato inconsistente.

4.5 Diagramma di Deployment

Nel diagramma di deployment abbiamo mostrato come gli artefatti software si dispongono tra i vari nodi e come comunicano tra loro.

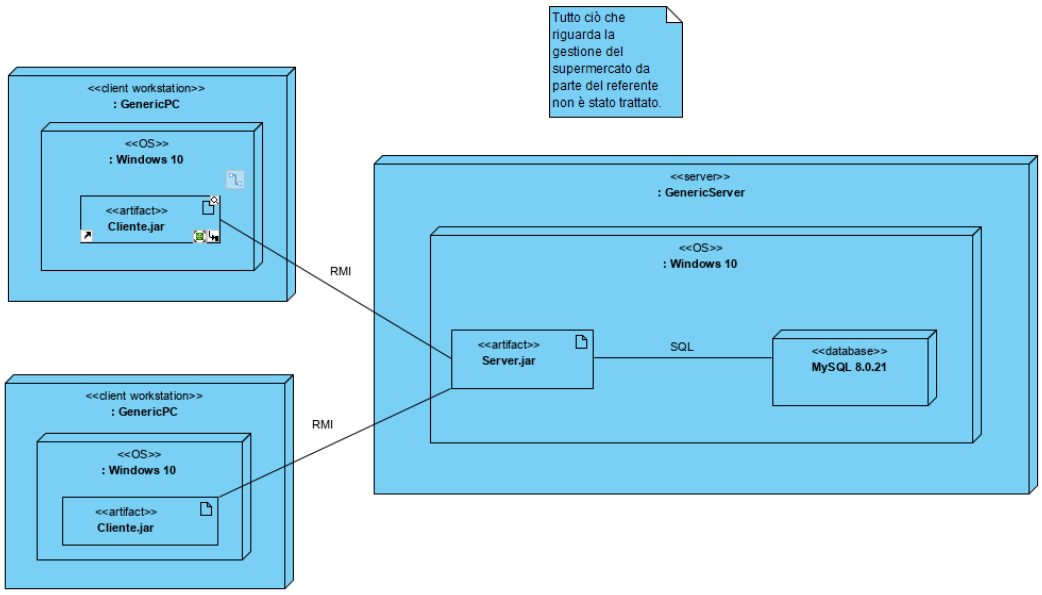

Figura 4.11: Deployment Diagram.